9. Tipos estándar

<tipos estándar> ::= 'integer' |

'real' |

'boolean' |

'char'

9.1. Enteros. Integer.

9.1.1. Valores.

Los valores de tipo entero son un subconjunto de los enteros. Se define el identificador maxint, que es estándar en todas las implementaciones de Pascal, y cuyo valor es el del máximo entero representable en el computador en que se está trabajando. Entonces el tipo entero es el conjunto de enteros, tales que:

$$|x| <= maxint$$

9.1.2. Operaciones aritméticas.

Los siguientes operadores dan un resultado de tipo entero, cuando son aplicados a operandos de tipo entero:

Multiplicación.	Operador *
Resta.	Operador -
Suma.	Operador +
División entera.	Operador DIV
Resto división.	Operador MOD

La operación da un resultado exacto, excepto en situaciones que provoquen rebalse.

- La división entera trunca el cuociente, no redondea:

Ejemplos:	15 div 4	= 3
	-15 div 4	= -3
	15 div(-4)	= -3

3 div 4 = 0

- El resto de la división entera, se define para operandos positivos y es tal que:

a mod b= a-((a div b)*b)

- Si a y b son de tipo entero, la operación es correcta si:

|a op b| <= maxint

- Las reglas de asociatividad, sólo son válidas cuando las asociaciones cumplen lo anterior.

- La inversión de signo se logra con la operación monádica signo menos:
Ejemplo: y := -x

9.1.3. Comparaciones.

Consisten en considerar la relación de orden entre dos variables de tipo entero. Los operadores relacionales son:

> mayor
< menor
= igual
>= mayor o igual
<= menor o igual
<> diferentes

Si los operandos son enteros, se obtiene un resultado de tipo Booleano.

9.1.4. Incrementos y decrementos.

Se dispone también de las funciones estándares:

succ(x) da el siguiente entero. Sucesor de x.
pred(x) da el antecesor de x.

Pero los programadores suelen preferir escribir los contadores mediante: x+1 y x-1 respectivamente.

9.1.5. Funciones estándares con resultado de tipo entero y argumento entero.

abs(x) da el valor absoluto de x.
sqr(x) da el cuadrado de x. (square en inglés).

9.1.6. Asignación en enteros.

A una variable entera sólo puede asignársele el valor de una expresión entera.

Si se desea emplear un operando real en un lugar donde sólo se aceptan valores de tipo entero, pueden usarse las funciones de conversión explícita:

a) Truncamiento:

trunc(x)

Si x es real, se trunca la parte fraccionaria de x.

Ejemplos:

trunc(3.01) = 3
trunc(3.99) = 3
trunc(-3.3) = -3
trunc(-3.8) = -3

b) Redondeo: se redondea al entero más próximo.

La función redondeo está definida según:

$$\text{round}(x)=\begin{cases}\text{trunc}(x+0.5) & \text{para } x>=0\\ -\text{trunc}(-x+0.5) & \text{para } x<0\end{cases}$$

Ejemplos:

round(3.45) = 3
round(3.5) = 4
round(-3.3) = -3
round(-3.8) = -4

Las funciones anteriores se denominan también de transferencia. Transfieren de real a entero.

La principal razón para distinguir entre números enteros y reales yace en la diferente representación interna. Las operaciones aritméticas son realizadas por secuencias de

instrucciones diferentes para cada tipo. Cuando se dice que se efectuá una conversión, se está realizando un cambio de la representación de punto flotante en la correspondiente representación interna de un entero.

Si bien en un sentido matemático los enteros son un subconjunto de los reales, en programación es preferible considerarlos disjuntos.

9.2. Reales. Real.

9.2.1. Valores.

El tipo real es un subconjunto de los números reales. El rango de representación depende del compilador con que se está trabajando. Debido a la precisión limitada de cualquier computador, las operaciones con tipo real son sólo una aproximación del valor que se obtendría si se trabajara con números reales.

En el eje real, por pequeño que sea un intervalo, siempre dentro de él existen infinitos números reales. El eje real es un continuo. Por el contrario, la representación del tipo real produce un número finito de números en un intervalo. En un caso extremo, cuando el intervalo esté formado por dos valores de tipo real, que difieren en el menor valor representable, no habrá ningún número de tipo real dentro del intervalo. El tipo real tiene magnitud y precisión limitadas.

Las operaciones peligrosas son las sumas y restas, particularmente la resta de dos valores casi iguales; en este caso se pierden cifras significativas. También la división por un número muy pequeño puede llevar fácilmente al rebalse.

Casi siempre pueden evitarse las divisiones por números muy pequeños.

Ejemplo:

```
abs(x/y)<delta;
```

puede escribirse:

```
abs(x)<delta*abs(y)
```

En Pascal, un operando que puede ser de tipo real, puede ser reemplazado (en el texto) por un operando de tipo entero. Es decir la conversión la efectúa automáticamente el compilador, sin necesidad de una función de transferencia explícita. Esto facilita escribir expresiones de tipo real que tengan información proveniente de variables enteras. Sin embargo, es preferible codificar sin mezclar tipos.

9.2.2. Operadores. Manipulación.

Están definidos la multiplicación *, la división /, la suma + y la resta -.

Si los operandos son reales el resultado es real. Si uno de los operandos es entero y el otro real, el resultado es real (conversión implícita). En el caso de la división ambos operandos pueden ser enteros y el resultado será de tipo real. Debe evitarse usar contadores de tipo real. Puede comprobarse que 10 veces 0.1 difícilmente será 1.0.

9.2.3. Funciones estándares.

Las siguientes funciones de argumento real dan un resultado real:

abs(x): valor absoluto de x
sqr(x): x al cuadrado

Las siguientes funciones pueden tener argumento entero (conversión implícita a real) o real, el resultado será de tipo real:

sin(x):	seno de x
cos(x):	coseno de x
arctan(x):	arcotangente de x
ln(x):	logaritmo natural de x
exp(x):	exponencial de x. {e^x}
sqrt(x):	raíz cuadrada de x

En las funciones trigonométricas x está en radianes.

No pueden aplicarse las funciones succ y pred (sucesor y predecesor) con argumento real.

9.2.4. Operadores relacionales.

El tipo real está equipado con los operadores relacionales que permiten establecer comparaciones entre valores de expresiones. El resultado será siempre de tipo booleano.

No es recomendable establecer condiciones absolutas trabajando con números de tipo real; esto debido a la inexactitud inherente a la representación. Por ejemplo: x=1.0E-5

Siempre será más confiable establecer condiciones relativas. Tampoco es recomendable establecer comparaciones de igualdad entre dos variables de tipo real.

9.2.5. Asignación en real.

A una variable real puede asignársele el valor de una expresión entera o real.

Internamente, la acción de asignación corresponde a copiar un valor obtenido desde una zona de la memoria en el espacio asignado a la variable. Si la expresión es de tipo entero, el compilador genera, en forma automática, una orden para cambiar la representación a punto flotante.

9.2.6. Funciones dependientes de la instalación.

Además de las funciones estándares es usual que en el manual del usuario del compilador con que se está trabajando, figuren una serie de funciones de argumento real. Su empleo debe restringirse pues limita la portabilidad de los programas desarrollados; es decir, dificulta el correr un programa, desarrollado en base a rasgos dependientes de un sistema, en otra instalación.

9.2.7. Sobre precisión finita en manipulación de reales.

Supongamos que disponemos de números reales que tengan 6 cifras decimales de precisión.

a) Errores de computación.

a1) Truncamiento:

Al evaluar la expresión siguiente:

(1.0/3.0)*3.0

Podría esperarse un resultado igual a 1.0; sin embargo al trabajar con 6 cifras, el resultado de la división será: 0.333333 y por lo tanto el valor de la expresión será: 0.999999.

a2) Rebalse.

Se tiene que: (10000.1-10000.0)+0.04=0.14

Pero (10000.1+0.04)-10000.0=0.10

Debido a que la suma se efectúa con 6 cifras, redondea 10000.14 a 10000.1. Nótese que con valores de tipo real no son válidas las reglas de asociatividad que se emplean con números reales.

b) Errores de Conversión.

b1) Algunos números racionales no pueden ser expresados como fracciones decimales con un número finito de cifras. Ejemplo de esto es 1/3 = 0.33333....

Al convertir este número a representación interna punto flotante se tendrá un error debido a la conversión.

b2) En forma similar hay números fraccionarios decimales que no pueden expresarse con un número finito de cifras fraccionarias binarias.

Ejemplo: 0.3 = 0.25 + 0.05
= 2^-2 + 0.03125 + 0.01875
= 2^-2 + 2^-5 + 0.015625 + 0.003125
= 2^-2 + 2^-5 + 2^-6 + 0.003125

Entonces con 6 cifras binarias, puede representarse 0.3 mediante:

0.010011

c) Errores de observación.

Externamente, cuando se efectúan mediciones físicas se tendrá errores de observación. Que dependen del instrumento y forma de medición.

Si un sistema tiene un número razonable de cifras significativas, en general los errores de observación serán mayores que los errores de conversión y de computación. Excepción de lo anterior es cuando, debido a un alto número de iteraciones, se producen acumulación de errores de computación; en este caso, los resultados de la computación son prácticamente inútiles.

d) Deben evitarse las comparaciones absolutas.

Supongamos que disponemos de tres variables de tipo real: a, b y c que representan los lados de un triángulo. Se desea marcar mediante la variable booleana, Pitágoras si el triángulo es o no rectángulo.

El segmento:

Pitágoras := sqr(a) = sqr(b)+sqr(c)

Difícilmente se cumplirá la exacta igualdad entre dos valores reales computados, debido a los errores descritos anteriormente.

Una versión más eficiente sería:

Pitágoras := abs((sqr(b)+sqr(c))/sqr(a)-1) < 0.0001

En la que se intenta probar que a^2 tenga 4 cifras significativas iguales a (b^2 + c^2). La tolerancia de 0.0001 debe escogerse de acuerdo a los errores de observación y de computación.

e) No deben emplearse variables de tipo real como contadores. El siguiente programa intenta generar números desde 1.0 a 1.1 en pasos de 0.001.

```
program contreal;
var x:real;
begin
  x:=1.0;
  while x<=1.1 do
  begin
    writeln(x:12:6); {escribe con 6 decimales en}
    x:=x+0.001      {campo de 12}
  end
end.
```

Debido a que la representación interna de 0.001 es aproximada, este valor repetidamente sumado a x origina una desviación creciente respecto del valor esperado. Mayor es el efecto sobre una función que dependa de x, y que estuviese dentro del lazo de repetición (produciría errores de computación).

Puede evitarse la acumulación de errores, usando un contador entero y calculando cada vez el valor de x.

```
program conint;
var i:integer;
    x:real;
begin
   i:=1;x:=1.0;
   while i<=100 do
   begin
     writeln(x:12:6);
      x:=1.0+i*0.001;
     i:=i+1
   end
end.
```

9.2.8. Errores en reales.

a) Error absoluto.

Si xa es una aproximación de x, se define el error absoluto según:

$$ea = |x - xa|$$

b) Error relativo.

Siempre que x sea diferente de cero, se define el valor relativo según:

$$er = ea/ |x|$$

c) Cotas de error.

Si se dispone de una representación con k dígitos decimales en la mantisa, el máximo error relativo será 10^(-k+1) en caso de truncamiento; y 5*10^(-k), en caso de redondeo.

d) Cifras significativas.

Dos números, x e y, coinciden en s dígitos significativos, si s es el mayor entero no negativo para el cual:

|x - y|/|x| < 5*10^(-s)

La comparación de cifras significativas es importante cuando los números son muy grandes o muy pequeños.

9.3. Tipo Booleano. Boolean.

9.3.1.- Valores

Un valor booleano es uno de los valores lógicos denotados por los identificadores estándares: false y true.

Como se verá más adelante este tipo se define con una relación de orden:

true > false

9.3.2. Operadores lógicos. Manipulación.

Se definen los 3 operadores: NOT, OR y AND; cuyos operandos deben ser de tipo booleano y con resultado de igual tipo. La siguiente tabla define los operadores y los valores de las variables p y q:

p	q	p AND q	p or q	NOT p
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Para una adecuada manipulación de expresiones lógicas pueden ocuparse las Reglas del Algebra de Boole. Las cuales pueden establecerse a partir de las definiciones anteriores. Se empleará el signo <-> para denotar la equivalencia.

9.3.2.1 Reglas de redundancia.

a) x AND FALSE <-> FALSE
b) x AND TRUE <-> x
c) x OR FALSE <-> x
d) x OR TRUE <-> TRUE
e) x AND x <-> x
f) x OR x <-> x

g) x AND (NOT x) <-> FALSE
h) x OR (NOT x) <-> TRUE

9.3.2.2 Regla de involución.

NOT (NOT x) <-> x

9.3.2.3 Reglas de conmutatividad

a) x AND y <-> y AND x
b) x OR y <-> y OR x

9.3.2.4. Reglas de asociatividad.

a) (x AND y) AND z <-> x AND (y AND z)
b) (x OR y) OR z <-> x OR (y OR z)

9.3.2.5 Reglas de distributividad.

a) (x AND y) OR z <-> (x OR z) AND (y OR z)
b) (x OR y) AND z <-> (x AND z) OR (y AND z)

9.3.2.6 Leyes de De Morgan.

a) NOT (x AND y) <-> (NOT x) OR (NOT y)
b) NOT (x OR y) <-> (NOT x) AND (NOT y)

Ejemplo:
Dada una tabla de verdad, se puede derivar una expresión en términos de los operadores NOT, AND, OR. Mediante las reglas del Algebra de Boole puede obtenerse la forma equivalente más simple de la expresión. Lo anterior puede aplicarse en caso de tener una serie de acciones condicionadas, y cuando se desea obtener una condición para cada acción.

Se tiene la siguiente tabla de verdad:

p	q	p <= q
false	false	true
false	true	true
true	false	false
true	true	true

La última tabla de la columna se ha construído considerando que true > false; y se ha aplicado el operador relacional mayor o igual.

De la tabla puede leerse que p<=q cuando:

i) p y q sean ambos falsos; o
ii) p sea falso y q verdadero; o
iii) p y q ambos verdaderos.

Lo cual puede escribirse:

p<=q <-> ((NOT p) AND (NOT q)) OR ((NOT p) AND q) OR (p AND q)

Aplicando distributividad:

p<=q <-> (NOT p) AND ((NOT q) OR q) OR (p AND q)

Aplicando redundancia:

p<=q <-> (NOT p) OR (p AND q)

Aplicando distributividad:

p<=q <-> ((NOT p) OR p) AND ((NOT p) OR q)

Nuevamente redundancia:

p<=q <-> (NOT p) OR q

Relación que se podría haber obtenido más simplemente, observando que p<=q es falso, sólo si q es false y p es true; es decir (p AND NOT q).

Entonces:

p<=q <-> NOT (p AND NOT q) <-> NOT p OR q

Debe notarse que la implicancia lógica, una de las 16 operaciones Booleanas entre 2 variables, puede escribirse en términos de los operadores básicos (NOT, OR, AND) o empleando operadores de relación asumiendo que se cumple la relación de orden false< true.

Entonces el operador implica puede leerse: menor o igual.

Así pueden definirse los operadores booleanos:

i) Equivalencia:

p=q <-> (p=>q) AND (q=>p)

Se lee p igual a q.

ii) OR exclusivo:

p<>q <-> (p AND (NOT q)) OR ((NOT p) AND q)

Se lee p distinto de q.

9.3.3 Operadores relacionales.

En el ejemplo anterior puede verse que los operadores de relación actuando sobre operandos booleanos permiten las diferentes operaciones booleanas entre dos variables.

Otro aspecto de interés es establecer las bases teóricas de los tipos ordinales. Es decir, aquellos en que está definido un orden. Para un elemento de este tipo, sólo hay un sucesor y sólo un antecesor.

Formalmente: Un tipo es ordinal si dados dos valores a y b pertenecientes al tipo:

i) Sólo una de las siguientes condiciones es verdadera:

a<b ; a=b; a>b

ii) Si a<b y b<c entonces: a<c

Puede comprobarse que mediante operadores lógicos NOT, OR y AND sólo es necesario definir el operador < ; el resto de los operadores relacionales pueden obtenerse en término de los operadores anteriores.

Se tienen:

a<=b	<-> NOT (b<a)
a=b	<-> (a<=b) AND (b<=a)
a<>b	<-> NOT (a=b)
a>=b	<-> b<=a
a>b	<-> b<a

Donde la expresión a la derecha puede considerarse la definición del operador de relación que figura a la izquierda.

Es por lo anterior que, al definir false < true, pudieron definirse los operadores relacionales con operandos booleanos.

9.3.2 Asignación a una variable booleana.

Sólo se acepta asignar un valor lógico a una variable booleana.

9.3.5. Condiciones relativas a desigualdades.

En el álgebra suelen describirse situaciones mediante desigualdades. Una de frecuente ocurrencia es:

x<y<z

Debido a la estructura de expresiones, en Pascal, la descripción de la desigualdad anterior se ve obscurecida. Debe plantearse, si x, y, z son de tipo entero:

(x<y) AND (y<z)

El empleo de paréntesis es obligatorio.

El ejemplo anterior ilustra como mediante los operadores de relación y los lógicos pueden escribirse desigualdades.

9.3.6. Entrada/Salida para tipo Boolean.

Debe considerarse que no pueden leerse, mediante la instrucción read, variables de tipo booleano.

La instrucción: write(<expresión booleana>)

Escribe las secuencias TRUE o FALSE, de acuerdo al valor de la expresión booleana.

La instrucción: write (<exp. booleana> : <exp. entera>)

Escribe las sentencias anteriores, en un campo de ancho dado por el valor de la expresión entera, justificadas por la derecha. Es decir, precedidas de espacios hasta completar el ancho.

9.4. Tipo caracter. Char.

9.4.1. Valores

Cualquier sistema de computación se comunica con el ambiente externo mediante algunos dispositivos de entrada y salida (input, output).

Estos dispositivos leen, escriben o imprimen los elementos de un conjunto fijo de caracteres. Este conjunto constituye el rango de valores del tipo CHAR.

Desgraciadamente, diferentes sistemas de computadores pueden emplear diversos conjuntos de caracteres, lo cual hace difícil y a veces tediosa la comunicación entre los sistemas. Se entiende por comunicación el intercambio de programas y datos.

Existe un conjunto estandarizado internacionalmente, denominado ISO. El ISO estándar define 128 caracteres; 33 de ellos se denominan caracteres de control, los restantes 95 se denominan caracteres gráficos o visibles.

El conjunto está ordenado y existe asociado a cada carácter un número entero de orden, que es el código.

El conjunto deja, sin embargo, algunos lugares abiertos. Estos pueden llenarse con diferentes caracteres de acuerdo a los deseos de cada país para establecer su propio estándar. El conjunto más empleado es el ISO estándar norteamericano denominado ASCII.

A continuación entregamos en forma de tabla el código ASCII, el número de orden de cada carácter se obtiene: sumando el número ubicado a la izquierda del renglón al que pertenece, con el número sobre la columna respectiva. Las dos primeras columnas y el carácter con número de orden 128 son caracteres de control; debido a que no son gráficos se denotan por abreviaciones que recuerdan sus usos habituales. Los caracteres de control se emplean principalmente para controlar funciones de los dispositivos, para delinear y estructurar textos (formato), y para controlar la comunicación entre dispositivo y computador (protocolos).

Como veremos más adelante, un importante papel de los caracteres de control es especificar el final de una línea, de una página, o de un texto. Sin embargo no hay un estándar universal para estos propósitos; denotaremos por eolm (end of line marker) al carácter, o secuencia de caracteres de control que se emplean para especificar el final de una línea. También emplearemos el símbolo eofm (end of file marker) para referirnos al carácter, o caracteres, de control que se usen para marcar el fin de un archivo. Los valores actuales dependerán del sistema que se esté empleando.

Código ASCII.

	0	16	32	48	64	80	96	112
0	nul	dle		0	@	P	`	p
1	soh	dc1	!	1	A	Q	a	q
2	stx	dc2	"	2	B	R	b	r
3	ext	dc3	#	3	C	S	c	s
4	eot	dc4	$	4	D	T	d	t
5	enq	nak	%	5	E	U	e	u
6	ack	syn	&	6	F	V	f	v
7	bel	etb	'	7	G	W	g	w
8	bs	can	(	8	H	X	h	x
9	ht	em	)	9	I	Y	i	y
10	lf	sub	*	:	J	Z	j	z
11	vt	esc	+	;	K	[	k	{
12	ff	fs	,	<	L	\	l	|
13	cr	gs	-	=	M	]	m	}
14	so	rs	.	>	N	^	n	~
15	si	us	/	?	O	_	o	del

Algunos caracteres de formato:

bs back space
ht horizontal tab (tab)
lf line feed
vt vertical tab
ff form feed
cr carriage return

Algunos caracteres de funciones:

bel campanilla
dc1 device code 1, también XON
dc3 device code 3, también XOFF

Los dos últimos suelen emplearse para detener y reanudar una ráfaga de caracteres desde el computador hacia el terminal. Estableciendo el conocido protocolo de software denominado XON/OFF.

Una forma usual de escribir los caracteres de control es mediante una secuencia de caracteres gráficos. Los de la primera columna se anotan con un sombrero ^ seguido del correspondiente carácter ubicado en la quinta columna. Para la segunda columna (16 a 31) la secuencia es el carácter ^ seguido del correspondiente de la sexta columna.

Esta notación es alternativa a la que figura en la tabla, y permite generar caracteres de control desde el teclado mediante la presión de la tecla control (CTRL) seguida del carácter; en forma similar a como se efectúa la generación de una letra mayúscula; dada por la secuencia (shift) seguida de la correspondiente letra. Debe mantenerse oprimida la tecla CTRL y luego se oprime la del carácter.

Ejemplos:

cr = ^M
Xoff = ^S

9.4.2. Manipulación

Los valores de tipo char, se anotan en el texto de un programa precediéndolos y antecediéndolos por un apóstrofe.

Ejemplos:

El espacio ' '
La a minúscula 'a'
El apóstrofe '''' ; se le anota dos veces.

Evidentemente no están definidas operaciones aritméticas para el tipo carácter (char).

a) Asignación.

Sólo puede asignarse a una variable de tipo carácter una expresión que al ser evaluada tenga un valor perteneciente al conjunto de caracteres.

Ejemplo: c:='?'

b) Funciones de Transferencia.

Las funciones estándar chr y ord establecen una relación entre el conjunto de caracteres y un subconjunto de los números naturales.

Es decir, entre un carácter y el número ordinal correspondiente (suma del valor renglón y columna en la tabla).

b1) De número a carácter. Chr.

El argumento es un ordinal (entre 0 y 127), el valor de la función es de tipo char.

Ejemplo: chr(65) tiene el valor 'A'
chr(i) tiene el valor del carácter asociado al ordinal i.
chr(13) es el carácter de control, identificado por cr (carriage return)

Esta función permite representar los caracteres de control.

b2) De carácter a número. Ord.

El argumento es de tipo carácter, y su valor es el número ordinal asociado.

Ejemplo: ord('A') tiene el valor 65

Nótese que los ordinales de las minúsculas son mayores (en 32) que sus correspondientes mayúsculas.

Los ordinales reflejan un ordenamiento alfabético para las letras y uno numérico para los dígitos.

b3) Propiedades.

i) Las funciones chr y ord son inversas.

chr(ord(ch)) Equivale a ch (ch de tipo carácter).

ord(chr(n)) Equivale a n (n es entero).

ii) La computación del valor numérico representado por un dígito ch (literal numérico o carácter numérico), puede lograrse con:

ord(ch)-ord('0')

Ejemplo:
ord('9')-ord('0') = 57-48 = 9

iii) La computación del dígito (carácter) que represente al valor numérico entero n, entre 0 y 127, se logra con:

chr(n+ord('0'))

Los dos puntos anteriores dependen de la adyacencia de los 10 dígitos en el conjunto de caracteres. Las dos fórmulas anteriores se usan típicamente en rutinas que convierten secuencias de dígitos (literales numéricos) en números y viceversa. Estos procesos deben efectuarse para convertir la secuencia de caracteres (dígitos) digitados en el teclado en un valor numérico que será asignado a una variable.

c) Relaciones de orden. Comparaciones.

El conjunto de caracteres es ordenado. Si c1 y c2 son variables de tipo char, se define la relación básica de orden mediante:

c1<c2 si y sólo sí ord(c1)<ord(c2)

Puede comprobarse, que la definición anterior, permite usar el resto de los operadores relacionales con operandos de tipo char.

En general si R representa a: =, <>, <=, >=, >, < se tienen:

c1 R c2 si y sólo sí ord(c1) R ord(c2)

Observando la tabla ASCII, el ordenamiento es alfabético:

Ejemplos:

'B'>'A' es verdadero
'A'<'a' tiene valor true
'2'>='1' tiene valor true
'z'<'a' tiene valor false

Si ch es de tipo char, las siguientes variables booleanas son ejemplos de expresiones útiles para la manipulación de caracteres:

```
minúscula := (ch>='a') and (ch<='z');
mayúscula :=( ch>='A') and (ch<='Z');
control := ch<=' ';   {espacio}
dígito := (ch>='0') and (ch<='9');
letra := mayúscula or minúscula;
gráfico := not control;
signo := gráfico and (not letra) and (not dígito);
```

Y también para convertir el valor de una variable entera en una secuencia de caracteres que serán enviados al terminal.

Si x es una variable de tipo entera y ch de tipo char, el siguiente segmento implementa la instrucción read(x). Se efectúa empleando la instrucción read(ch), que se supone implementada mediante una secuencia de instrucciones de máquina.

```
...
x:=0;read(ch);
while ('0'<=ch) and (ch<='9') do
begin
  x:=10*x+(ord(ch)-ord('0'));
  read(ch)
end
```

El segmento implementa la definición léxica:

<número entero> ::= <dígito> {dígito}

Recordar que es equivalente a :

<número entero> ::= {dígito *}

La condición de término es leer un carácter que no sea numérico; en cuyo caso la variable ch toma el valor del último símbolo digitado. Segmentos como el anterior se denominan analizadores léxicos (scanner) o reconocedores; y son máquinas secuenciales que son estudiadas en detalle en textos de Sistemas Digitales y Autómatas.

Si se desea convertir una letra minúscula en mayúscula, puede emplearse:

```
if minúscula then ch:=chr(ord(ch)-32)
```

Una mejor codificación es:

if minúscula then ch:=chr(ord(ch)-ord('a')+ord('A'))

d) Funciones estándar de orden.

Están definidas las funciones con argumento de tipo carácter y cuyo resultado es de igual tipo, que permiten obtener el sucesor (succ) y antecesor (pred) de un carácter.

En el caso que existan, se definen:

pred(ch) equivale a chr(ord(ch)-1)
succ(ch) equivale a chr(ord(ch)+1)

Ejemplos: pred('A') tiene el valor '@'
succ(chr(127)) no está definido.

9.4.3. Entrada. Salida.

a) Salida.

Si e es una expresión de tipo char: write(e), significa escribir el valor de e en el terminal; no se escriben los apóstrofes.

Ejemplo: write('a') al ser ejecutada escribe la letra a.

Si m es una expresión entera: write(e:m)

Escribe m-1 espacios seguidos del carácter que es el valor de e.

b) Entrada.

i) Si v es de tipo char: read(v) puede interpretarse en tiempo de ejecución como la espera hasta digitar un carácter, que será asignado a v.

ii) Debe recordarse que la ocurrencia de un readln se interpreta como el proceso de descartar todos los caracteres que se digiten, hasta encontrar un eolm (marcador de fin de línea).

iii) Entonces: readln(v) es equivalente a asignar a v el primer carácter que se digite; y el posterior descarte de todos los caracteres que se escriban hasta el eolm.

Es decir equivale a la secuencia: read(v); readln

9.4.4. Aplicaciones empleando tipo char.

a) Proceso de caracteres.

Se asume que se escribirá en una línea del terminal una secuencia de caracteres con un nombre y un apellido. Pueden existir espacios antes, después y entre las palabras. Obviamente debe existir un espacio, por lo menos, entre palabras.

Se desea escribir, en el comienzo de una línea, la inicial del apellido seguida de punto, y luego de un espacio la inicial del nombre seguida de un punto.

El siguiente programa es un ejemplo de procesamiento no numérico.

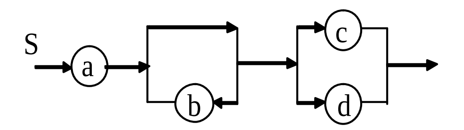
```
Program iniciales;
    var ch, ninicial,napellido:char;
    begin
      read(ch);                  {inicia ch}
      while ch=' ' do read(ch);   {descarta espacios}
      ninicial:=ch;
      while ch<>' 'do read(ch); {salta restodel nombre}
      while ch=' ' do read(ch);  {salta blancosantes del apellido}
      napellido:=ch;
      while ch<>' ' do read(ch);  {salta resto del apellido}
      readln;writeln;
      writeln(napellido,'. ',ninicial,'.')
    end.
```

En el programa se emplea la técnica de análisis de leer un carácter por adelantado y procesar de acuerdo a su valor (lookahead).

La lectura de un carácter, sin efectuar una asignación el valor adquirido es equivalente a descartarlo. Al leer nuevamente se borra (se pierde) el valor anterior; se escribe 'encima' del anterior.

b) Reconocedor de expresiones regulares.

Se desea saber si una secuencia de caracteres digitados cumple determinada sintaxis S.



Ejemplos de secuencias L que cumplen S:

ac ad abc abd abbc abbd .....

Las sentencias anteriores muestran cierta regularidad en su estructura; por esto se dice que S es una expresión regular.

El vocabulario, en este caso, es V=(a,b,c,d). Nuestro problema consiste en: dada una expresión regular S que define un conjunto de secuencias sobre V; y dada una secuencia L de símbolos sobre V, se desea obtener un algoritmo que determine si L pertenece a S. Esto se denomina reconocedor de una secuencia.

```
     ... (*  <S>::='a'{'b'}('c'|'d')  *)
   read(ch);  {se lee por adelantado}
     gol:=false;
     if ch='a'
   then
      begin
read(ch);
         while ch='b' do read(ch);
         gol:=(ch='c') or (ch='d')
         end;
   ...  {La variable gol es verdadera si L pertenece a S.}
```

9.5 Ejercicios.

9.5.1.- Evaluar las siguientes expresiones aritméticas.

a) 2*3 +4*5
b) 2 * 3+4* 5
c) 15 DIV 4*4
d) 15 DIV (4*4)
e) 2+3*4-5
f) 6.25/1.25+1.5
g) 3+5<>5+3
h) 80/5/3
i) 4/2*3
j) sqrt(sqr(3)+11*5)
h) -x+y/z=(-x)+(y/z)

9.5.2. Determinar a que categoría sintáctica pertenecen.

a) (x+y+z)
b) not p
c) p or q
d) (x<=y) and (y<z)
e) x+y
f) i*j+1
g) x=1.5
h) p<=q
i) x-y*z-w=(x-(y/z))-w
j) exp(y*ln(x))

9.5.3. Escribir correctamente las siguientes líneas.

a) raiz:=-b+sqrt(b**2-4ac)/2a

b) $$x := \frac{\frac{1}{a}+\frac{1}{b}}{\frac{1}{c}*\frac{1}{d}}$$

9.5.4. Simplificar:
a) (a OR x) AND (b OR x) AND (c OR x)
b) a AND c OR b AND c OR c ; poner paréntesis.

9.5.5. Reconocer: a) {'a'{'b'}'c'}'d'

b) 'a'{'b'('c'|'d')}'e'

9.6. Ejemplos.

9.6.1. Se tienen tres variables booleanas a,b,c.

a:=false; b:=true; c:=true;

Determinar el valor de b, para las siguientes asignaciones:

a) b:= (a <> c) OR (a >= c)
b) b:= a OR NOT b OR c
c) b:= (a < b) AND (c OR a)

Solución

a) a <> c <-> false <> true <-> true
a >= c <-> false >= true <-> false

b:= true OR false <-> true

b) El operador lógico de mayor jerarquía es NOT; por lo tanto se evalúa primero.

NOT b <-> NOT true <-> false

Entonces:
b: a OR false OR c;

Evaluando el primer OR :

a OR false <-> false OR false <-> false;

Finalmente :
false OR c <-> false OR true <-> true;

b:= a OR NOT b OR c <-> true

c) a < b <-> false < true <-> true

c OR a <-> true OR false <-> true

b:= true AND true <-> true

9.6.2. Determinar el valor de las condiciones:

a) ord('A') > ord('a')
b) succ('B') <> pred('D')
c) '3' = chr(3)

Solución

a) ORD('A') = 65
ORD('a') = 97

Lo cual es debido a que, en el código ASCII, las mayúsculas están antes que las minúsculas; así también sus ordinales.

Como 65 no es mayor que 97, se tiene que la condición a) tiene valor false.

b) SUCC('B') = 'C'
PRED('D') = 'C'

Entonces : 'C' <> 'C' tiene valor false

c) Es verdadero ssi, el terecer caracter del ASCII es '3', sino es falso.
Pero, el tercer caracter corresponde a un caracter de control.

ORD(CHR(3)) = 3
ORD('3') =51

La condición 51 = 3 toma valor false.

9.6.3. Determinar el tipo de las variables:

a) x:= 3.0 + 3
b) y:= abs(x) + 1.0E-5
c) y:= sen(x) + 3 div 2

Solución:

a) 3.0 es real ; 3 es entero
La suma de un real y un entero debe asignarse a una variable de tipo real (x), ya que se realiza una conversión implícita del entero a real.

b) Si x es real --> abs(x) es real
Si x es entero --> abs(x) es entero
1.0E-5 es real, lo que indica que la suma de dos reales es real (si x es real) y debe asignarse a una variable de tipo real (y).

Si x es entero se está en la misma situación que en a), por lo tanto, y debe ser de tipo real.

c) sen(x) acepta argumentos de tipo real o entero, pero arroja resultado real.

3 DIV 2 arroja un resultado entero.

Como el resultado de la suma de un real y un entero es real; la variable y debe ser de tipo real.

9.6.4. Potencias de Enteros.

Pascal estándar no tiene definida la función potencia de un número entero.

Dados x e i se desea obtener : z=x^i

En lenguaje Fortran se dispone de la potencia de enteros y se anota : z=x**i

a) Una primera implementación, puede ser :

```
z:=1;
while i>0 do z:=z*x; i:=i-1 end;
```

Donde se advierte con claridad el algoritmo.

Pero el número de iteraciones es elevado. Se busca un método que sea más eficiente.

b)
```
z:=1;
while i>0 do
  begin
    while not odd(i) do
      begin
```

```
      x:=sqr(x); i:=i div 2
    end;
  i:=i-1; z:=z*x
end;
```

Si i es par, se reemplaza x por x al cuadrado; e i se reduce a la mitad. Cuando i es impar, sólo se actualiza z; en este caso es el mismo algoritmo visto en a).

c) El mismo algoritmo, visto en b), puede plantease más simplemente. Observando que un número impar dividido por dos, efectúa un truncamiento que es equivalente a restar uno.

Puede escribirse :

```
z:=1;
while i>0 do
  begin
    if odd(i) then z:=z*x;
    x:=x*x; i:=i div 2
  end;
```

Modificar el algoritmo para obtener el máximo común divisor de dos enteros x,y ; para obtener el mínimo común múltiplo de esos enteros.

Debe considerarse que :

mcd(x,y) * mcm(x,y) = x*y

9.6.5 Cálculo de raíces de polinomios.

En términos matemáticos el problema puede plantearse: Dada una función f(x), continua en un intervalo a<=x<=b, y conociendo que f(a) < 0 < f(b), se desea encontrar un valor de x, perteneciente al intervalo, tal que :

f(xr) = 0

Se dice que xr es una raíz del polinomio f(x).

En forma gráfica:

f(x)
xr
a
x
b

Si se subdivide el intervalo en pequeños segmentos de igual largo, podemos ir evaluando la función a partir de a; en algún momento debe cambiar el signo de la función. En ese instante se habrá aislado la raíz de f(x), en un intervalo menor.

Si se desea encontrar la posición del cero, con mayor precisión, se debe repetir el proceso anterior con un menor intervalo.

Es decir :

```
x:=a; i:=0;
repetir
  i:=i+1;
  x:=a+i*intervalo;
  evaluar f(x);
hasta f(x)>0 y x<b
```

Al salir del lazo, la raíz está confinada en:

a + (i-1)*intervalo < xr < a + i*intervalo

La elección del tamaño del intervalo incide con el tiempo de procesamiento para encontrar la raíz. Una alternativa es escoger un intervalo muy pequeño, lo que permite aislar un valor de la raíz con la precisión que se desee. Sin embargo, el algoritmo será de lenta ejecución, y se deben realizar innumerables computaciones que no aportan nada al resultado final. Debe notarse la similaridad de este caso con la búsqueda lineal de un valor en un arreglo.

Si por el contrario, escogemos el intervalo igual a la mitad de (b-a), y cada vez se subdivide el nuevo intervalo en la mitad del anterior, se tendrá que después de n iteraciones, la raíz estará confinada en un intervalo de largo:

(a-b) * 2^(-n)

Si consideramos que un número real se representa como una secuencia de unos y ceros, en sistema binario; y que la división por dos significa un desplazamiento hacia la derecha de la secuencia, podemos establecer que en cada bisección se obtiene un bit del resultado.

También debe notarse la analogía del método anterior con la búsqueda binaria.

Puede demostrarse que la división del intervalo en dos es la elección óptima. En este método, la única información que se usa de la función es su signo; existen mejores métodos cuando se emplea el valor de la función (método de la secante y otros).

El algoritmo que subdivide en dos el intervalo, se denomina de bisección. En él, un valor de interés es el número de veces que se repetirá la división; como se dijo antes existe un número máximo de veces que se puede repetir la bisección. Este valor está en directa relación con el número de bits que se empleen para representar un real.

```
program biseccion(input,output);
const
   maxveces = 24;
   tolerancia = 1.0E-8;
var
   veces : integer;
   inferior, medio, superior : real;
   f : real;
begin
  read(inferior, superior);
  veces:=0;
  repeat
    medio:=(inferior + superior)/2.0;
    f:= ... {función evaluada con x:=medio};
    if f > 0
    then  superior:=medio
    else  inferior:=medio;
    veces:=veces+1
  until (abs(f)<tolerancia) or (veces>=maxveces);
  write ('raiz= ',medio)
end.
```

- Si por ejemplo la función es $f(x) = 3x^2 - 2x - 5$, la expresión deberá escribirse:

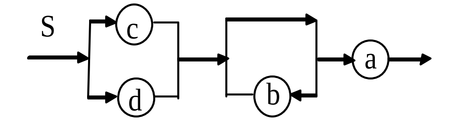
```
f := 3*sqr(medio) - 2*medio - 5
```

- Analizar si el algoritmo funciona propiamente si existen varias raíces en el intervalo.

- Analizar el significado de detener la repetición cuando:

(veces>=maxveces) or(abs(superior-inferior)>abs(a)*1E-8)

9.6.6. Dado el siguiente grafo sintáctico:



a) Escribir producción para S en BNF
b) Dar 5 ejemplos de sentencias que cumplan S
c) Escribir un programa que reconozca si una secuencia pertenece o no a S.

Solución

a) 'c' | 'd' es una alternativa.
'b' es una repetición.

S es la concatenación de lo anterior con 'a' :

<S> ::= ('c' | 'd') {'b'} 'a'

b) ca
cba
cbba
da
dba ...

c)
```
PROGRAM RECONOCEDOR;
VAR GOL : BOOLEAN; {determina si la secuencia es válida o no}
      CAR : CHAR;          {para leer carácter a carácter}
BEGIN
   GOL:=false;
   READ(CAR);      {lookahead}
   IF (CAR='c') OR (CAR='d')
      THEN
      BEGIN
         REPEAT    READ(CAR)  UNTIL CAR <> 'b';
         IF CAR='a' THEN GOL:=true
      END;
   IF GOL
      THEN WRITE('PERTENECE A S ')
      ELSE WRITE('NO PERTENECE A S ')
END.
```